# 5 Definitions

## Overview

This section of the Signalogic DSP Software Reference Guide documents structures defined for use in application-specific C/C++ and MATLAB programs when calling functions in DSPower series DLLs. The actual (and current) definition of these structures can be found in the include files noted for each. Example include hwlib.h, hwmgr.h, etc.

### Structures Used in DSPower Series Software

HWLISTENTRY

CONVERSIONINFO

ERRORMSGINFO

DSPDATARDYMSGINFO

## 5.1 Structure Definitions

There are several structure definitions used by DSPower series software. These are typically found in include files such as enmgr.h, hwmgr.h, hwlib.h, and splib.h. The sections below describe these structure definitions.

# 5.1.1 HWLISTENTRY Structure

Below is a description of the HWLISTENTRY structure, which is used in Hardware Manager calls such as DSGetHWMgrEntry and DSSetHWMgrEntry. The current definition of this structure is located in the DSPower-HWLib include file **hwmgr.h**.

The description below is divided into the following sections:

- information maintained in the Hardware Setup File
- information maintained internally by Hardware Manager, and is not stored in the Hardware Setup File

Each element in the structure description below includes a brief comment. See also the notes after the structure definition, and Signalogic DSP Software Users Guide section 8.3.1, Hardware Setup File Fields.

typedef struct {

| // type | // element | // comment |
|---|---|---|
| DESCSTR | szMfg; | // manufacturer field |
| DESCSTR | szDescription; | // description field: name/model, DSP type, etc. |
| DESCSTR | szBoardDesignator; | // board designator field |
| short int | MajorDriverID; | // major DSP/analog hardware driver ID field |
| short int | MinorDriverID; | // minor DSP/analog hardware driver ID field |
| char | szCodeFile[144]; | // DSP program file field (current executable code filename) |
| char | szCodeFileDef[144]; | // default DSP program file |
| VALSTR | szIOBaseAddr; | // current I/O base address field (in hex) |
| VALSTR | szIOBaseAddrDef; | // default I/O base address |
| VALSTR | szMemBaseAddr; | // current memory base address field (in hex) |
| VALSTR | szMemBaseAddrDef; | // default base memory address |
| short int | BusType; | // bus type field |
| short int | BusTypeDef; | // default bus type |
| VALSTR | szProcClock; | // processor clock field (specified in MHz) |
| VALSTR | szProcClockDef; | // default processor clock |
| VALSTR | szVoltageRanges; | // voltage ranges field (should be entered as 'input, output') |
| VALSTR | szVoltageRangesDef; | // default voltage ranges |
| short int | AnalClock; | // analog clock type field: 0 = external clock, 1 = fixed internal, 2 = prog internal |

```
short int       AnalClockDef;       // default analog clock type
VALSTR          szMemArch;          // memory architecture field: "L" = linear, "H" =
                                    modified Harvard, "V" = vector
VALSTR          szMemArchDef;       // default memory architecture
short int       NumProcessors;      // number of processors field
short int       NumProcessorsDef;   // default number of processors

// following parameters not stored in Hardware Setup File
// type         // element          // comment
short int       CallClass;          // calls-allowed classification
short int       Flags;              // internal information used to mark pending
                                    updates, deletes, etc.
} HWLISTENTRY;
typedef HWLISTENTRY FAR* PHWLISTENTRY;
```

## Notes

1) HWLISTENTRY structures should be completely initialized to zero before using, so default values can take effect for parameters not used. This can easily be accomplished using a call such as

```
_fmemset(&HWListEntry, 0, sizeof(HWLISTENTRY));
```

2) Supporting type definitions used above:

```
typedef char DESCSTR[50];
typedef char VALSTR[10];
```

## Example

## See Also

DSSetHWMgrEntry

DSGetHWMgrEntry

## Related Topics

Hardware Setup File Access

Selecting DSP/Analog Hardware Types

DSP/Analog Hardware Configuration

Hardware Values and Variables

## General Topics

Hardware Manager Access

Hardware Selector

Hardware Setup File

Error Handling in Hardware Manager

# HWLISTENTRY Example

Below is an excerpt from the dscope.cpp source code example file included with DSPower-HWLib software. It shows a typical include file section which allows compilation of applications using the DSPower-HWLib software. Remember that referring to the prototypes inside the DSPower-HWLib DLL include files shown below will cause the necessary .lib (library) files to be needed in the application's project file at link-time; for more information on DSPower-HWLib and DSPower-SPLib .lib files, see section 4.2.1, .LIB Files for DSPower-HWLib and DSPower-SPLib.

```
// typical application-specific include files
#include <windows.h>
#include <stdlib.h>
#include <math.h>

// DSPower-HWLib DLL include files
#include "enmgr.h"   // Engine Manager
#include "hwmgr.h"   // Hardware Manager
#include "hwlib.h"   // Hardware Library

// more Signalogic DLL include files, but used only for user-interface in demo
#include "toollib.h"
#include "helplib.h"
#include "minmax.h"  // local versions of min/max, if needed
```

### See Also

DSSetHWMgrEntry

DSGetHWMgrEntry

### Related Topics

Hardware Setup File Access

Selecting DSP/Analog Hardware Types

DSP/Analog Hardware Configuration

Hardware Values and Variables

### General Topics

Hardware Manager Access

Hardware Selector

Hardware Setup File

Error Handling in Hardware Manager

## 5.1.2 CONVERSIONINFO Structure

Below is a description of the CONVERSIONINFO structure, which is used in high-level DSPower-HWLib calls such as DSAcquireWvfrmFile and DSGenerateWvfrmFile. The current definition of this structure is located in the DSPower-HWLib include file **hwlib.h**.

The reprint below is partially divided into the following sections:

- basic acquisition/generation information
- real-time digital filter information
- trigger information
- stimulus & response (simultaneous input/output information)

Each element in the structure description below includes a brief comment. For more detailed information, see the additional explanations of some elements after the structure.

typedef struct {

**// basic acquisition/generation information**

| **// type** | **// element** | **// comment** |
|---|---|---|
| char | szFilename[144]; | // [path]filename of file being acquired/generated (see notes below) |
| short int | NumChan; | // number of channels |

| | | |
|---|---|---|
| short int | ChanList[MAXCHAN]; | // channel list, maps physical channels to waveform traces (entries should not exceed NumChan) |
| float | Fs; | // sampling rate (in Hz) |
| DWORD | NumSamples; | // number of samples to acquire/generate |
| char | szGainList[41]; | // string specifying gain list for input channels; can contain ',' and '-' other characters to specify lists and ranges (see notes below) |
| long | Offset; | // offset added to data (in A/D counts, can be +/-) |
| BOOL | DALooping; | // enables/disables continuous D/A looping; 0 or 1, valid only during generation (output) |
| BOOL | Monitor; | // enables/disables D/A loopback when recording; 0 or 1, valid only during acquisition (input) |
| BOOL | FileSplit; | // 1 enables A/D file-splitting for multichannel, 0 disables |
| BOOL | StoreOnAbort; | // 1 causes current waveform data to be stored if a user- or program-induced abort occurs |
| DWORD | Frmsiz; | // framesize (stored in waveform header; used for post-processing only) |
| **// real-time digital filter information** | | |
| **// type** | **// element** | **// brief description** |
| char | szFilter1[13]; | // filename of real-time digital filter file; can contain '/' character |
| char | szFilter2[13]; | // to allow dual-filter entry (filter files must be located on same path as waveform) |
| **// trigger information** | | |
| **// type** | **// element** | **// brief description** |
| WORD | TrigMode; | // trigger mode (0 = internal, 1 = external) |
| long | TrigLevel; | // trigger level (in A/D counts, can be +/-) |
| long | TrigDelay; | // delay from trigger relative to data (in sec, can be ±) |
| short int | NumTrigChan; | // number of trigger channels |
| short int | TrigChanList[MAXCHAN]; | // trigger channel list (entries should not exceed NumTrigChan |
| **// stimulus & response information** | | |
| **// type** | **// element** | **// brief description** |
| char | szStimName[13]; | // name of stimulus output file; used only for DSStimRespMeasure function (must be located on same path as waveform) |
| WORD | StimMode; | // controls stimulus usage in |

```
                                        Stimulus/Response function
long        StimDelay;                  // delay from stimulus onset to start of
                                        record (+/- value)
} CONVERSIONINFO;

typedef CONVERSIONINFO FAR* PCONVERSIONINFO;
```

## Notes

1) The entire CONVERSIONINFO structure should be initialized to zero before using, so that default values can take effect for parameters not used. This can easily be accomplished using a call such as

```
_fmemset(&ConversionInfo, 0, sizeof(CONVERSIONINFO));
```

2) Previous value used in either DSEngineOpen or DSSetEngineWaveformPath call is used for waveform filenames, unless explicit path information is found in the filename above.

***NOTE: For operation when a Hypersignal-Macro series program is the DSP Engine, szFilename above should not contain path information. The path should instead be set using the initial DSEngineOpen call, or a subsequent DSSetEngineWaveformPath call.***

3) The szGainList parameter is a string because input (or output) gain and/or attenuation can be specified as a software digital scaling value, or as a hardware gain value in Volts or dB, depending on the DSP/analog hardware type. Also, entry can be channel-specific, or specified as a channel-range.

4) TrigDelay parameter not yet operational.

5) PCONVERSIONINFO can be used to declare pointers to CONVERSIONINFO structures. For examples, see Example C/C++ Code: Playing Back Waveform Files, and Example C/C++ Code: Acquiring Waveform Files, in Signalogic DSP Software Users Guide section 4.2.2, Example Code: DSPower-HWLib and DSPower-SPLib Software.

## Example

## See Also

DSAcquireWvfrmFile

DSGenerateWvfrmFile

### Related Topics

Example C/C++ Code:  Acquiring Waveform Files

Example C/C++ Code:  Playing Back Waveform Files

### General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

## CONVERSIONINFO Example

Below is a partial excerpt from the dtape.cpp example source code file included with DSPower-HWLib software showing typical setup and initialization of a CONVERSIONINFO structure for use in a DSAcquireWvfrmFile call.  The dtape.cpp demo attempts to open Hypersignal-Macro or Hypersignal-Acoustic software as a DSP Engine, but the source code example below applies regardless of the engine.  One exception may occur in the szFilename parameter of the CONVERSIONINFO structure; for more information, see notes in section 5.1.2 above, CONVERSIONINFO Structure.

```
// create CONVERSIONINFO object memory; use zero-initialization to set most
parameters to default values

hConvInfo = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE |
LMEM_DISCARDABLE, sizeof(CONVERSIONINFO));

pConvInfo = (PCONVERSIONINFO)LocalLock(hConvInfo);

lstrcpy(pConvInfo->szFilename, szWaveformFilename);  // file to acquire

// setup for one channel, init channel list (example is sequential)
```

```
pConvInfo->NumChan = 1;          // one channel
for (j=0; j<pConvInfo->NumChan; j++) pConvInfo->ChanList[j] = j;

// setup up sampling rate, amount, gain, trigger information

pConvInfo->Fs =22050;  // 22.05 kHz sampling rate
pConvInfo->NumSamples = 100000L;  // 100,000 samples
lstrcpy(pConvInfo->szGainList, "1");  // see notes in CONVERSIONINFO
pConvInfo->NumTrigChan = 1;  // one trigger channel
pConvInfo->TrigChanList[0] = 0;  // freerun trigger

// start acquisition

DSAcquireWvfrmFile(hBoard, pConvInfo);

// free and delete memory

LocalUnlock(hConvInfo);
LocalFree(hConvInfo);
```

## See Also

DSAcquireWvfrmFile

DSGenerateWvfrmFile

## Related Topics

Example C/C++ Code:  Acquiring Waveform Files

Example C/C++ Code:  Playing Back Waveform Files

## General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

# 5.1.3 DSPDATARDYMSGINFO Structure

Below is a description of the DSPDATARDYMSGINFO structure, which pointed to by the lParam parameter of DSP Engine callback messages such as WM_DSPENGINE_BUFRDY and WM_DSPENGINE_FLGRDY sent by Engine Manager. The current definition of this structure is located in the DSPower-HWLib include file **enmgr.h**.

Each element in the structure description below includes a brief comment. See also the notes below, after the structure definition, and Signalogic DSP Software Users Guide section Buffer Ready Messages from DSP Engines and Semaphore Messages from DSP Engines, under section 7.4, Registering Callback Messages from DSP Engines.

```
typedef struct {
// type         // element          // comment
short int       nNum;               // buffer or flag number
DWORD           dwFlagValue;        // flag value
DWORD           dwFlagAddr;         // flag address
HGLOBAL         hBoard;             // board handle
} DSPDATARDYMSGINFO;
typedef DSPDATARDYMSGINFO FAR* PDSPDATARDYMSGINFO;
```

## Notes

1) The buffer or flag number reflects the original nBufferNum or nFlagNum values in used in DSWaitForBuffer or DSWaitForFlag calls.

2) The lParam value of the WM_DSPENGINE_BUFRDY and WM_DSPENGINE_FLGRDY messages points to a DSPDATARDYMSGINFO structure.

3) The wParam value of the WM_DSPENGINE_BUFRDY and WM_DSPENGINE_FLGRDY messages is the handle of the engine to which the DSP board buffer or flag was assigned.

## Example

## See Also

DSWaitForBuffer

DSWaitForFlag

DSRegisterEngineMsgWnd

WM_DSPENGINE_BUFRDY

WM_DSPENGINE_FLGRDY

## Related Users Guide Topics

Registering Callback Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

# DSPDATARDYMSGINFO Example

Below is an excerpt from the dscope.cpp source code example file (included with DSPower-HWLib software) showing a WM_DSPENGINE_BUFRDY message being received:

```
case WM_DSPENGINE_BUFRDY:  // buffer ready message sent by DSP engine

    pBuf = (int far*)GlobalLock(hBuf);  // lock memory to be used

    // transfer below illustrates two common type of DSP memory architectures

    if (uMemArch == DS_GMA_VECTOR) {  // vector data memory

        if (nCurBuf == 0)  status = DSGetMem(hBoard,
        DS_GM_VECTOR_DATA_X, dwTimBufferBaseAddr, DS_GM_SIZE16,
        pBuf, wBuflen);

        else  status = DSGetMem(hBoard, DS_GM_VECTOR_DATA_Y,
        dwTimBufferBaseAddr, DS_GM_SIZE16, pBuf, wBuflen);
    }
    else { // linear data memory

        status = DSGetMem(hBoard, DS_GM_LINEAR_DATA_RT,
        TimBufferBaseAddr+nCurBuf*Buflen, DS_GM_SIZE16, pBuf, wBuflen);
    }

    GlobalUnlock(hBuf);  // let memory float
```

```
if (!status) MessageBox(NULL, "DSGetMem problem", "Test Prog", MB_OK);

// switch buffers and write new buffer number

nCurBuf ^= 1;
DSPutEngineVarMem(hBoard, DSP_HOSTBUFNUM, nCurBuf);

// can we have another please, Sir..
DSWaitForBuffer(hBoard, nCurBuf, NULL, DS_WFB_POLLED);
break;
```

## Notes

1) A great deal of initialization is necessary before buffer ready and semaphore messages can be processed. The example above leaves out much of this; it is strongly recommended that the entire dscope.cpp file be examined before implementing code similar to the above example.

2) uMemArch is assumed to have been initialized beforehand using the DSGetMemArch call.

3) hBuf is assumed to have been created via a GlobalAlloc call, using a sizeof parameter equal to 2*wBuflen. Memory pointers passed to DSGetMem and DSPutMem calls *must not point to memory segments less than the product of the dwCount and DS_GM_SIZExx parameters* used in the corresponding DSGetMem or DSPutMem calls. This product is in bytes.

### See Also

DSWaitForBuffer

DSWaitForFlag

DSRegisterEngineMsgWnd

WM_DSPENGINE_BUFRDY

WM_DSPENGINE_FLGRDY

### Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

# 5.1.4 ERRORMSGINFO Structure

Below is a description of the ERRORMSGINFO structure, which is pointed to by the lParam parameter in DSP Engine status messages such as WM_DSPENGINE_COMPILEERROR and WM_DSPENGINE_RUNTIMEERROR sent by Engine Manager. The current definition of this structure is located in the DSPower-HWLib include file **enmgr.h**.

Each element in the structure description below includes a brief comment. See also the notes below, after the structure definition, and Signalogic DSP Software Users Guide section Status Messages from DSP Engines, under section 7.4, Registering Callback Messages from DSP Engines.

```
typedef struct {
// type          // element              // comment
short int        nErrorNum;             // error number or type
char             szErrorStatus[256];    // error/status string or error log filename
} ERRORMSGINFO;
typedef ERRORMSGINFO FAR* PERRORMSGINFO;
```

## Notes

1) The nErrorNum parameter can be interpreted as shown below:

| Value | Meaning |
|---|---|
| >= 0 | error number; szErrorStatus contains error condition string |
| < 0 | indicates szEngineError contains pathname of error log file |

2) Error log files are typically used in cases where the engine has found more than one error or warning.

3) The lParam value of the WM_DSPENGINE_xxxxERR messages points to an ERRORMSGINFO structure.

4) The wParam value of the WM_DSPENGINE_xxxxERR messages is the handle of the engine encountering the error.

## Example

## See Also

DSRegisterEngineMsgWnd

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_FUNCTIONERROR

WM_DSPENGINE_RUNTIMEERROR

## Related Users Guide Topics

Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

# ERRORMSGINFO Example

Below is a partial source code example showing a WM_DSPENGINE_COMPILEERROR message being received:

```
char szErrorStatus[256];
char szLogFile[256];
HFILE hFile;
OFSTRUCT ofstruct;

case WM_DSPENGINE_COMPILEERROR:

    int nErrorNum = (PERRORMSGINFO)lParam->nErrorNum;
    if (nErrorNum >= 0) {

        _fstrcpy(szErrorStatus, (PERRORMSGINFO)lParam->szErrorStatus);

        _fstrcat(szErrorStatus, "; Error Number = ");
        char tmpstr[30];
        itoa(nErrorNum, tmpstr, 10);
        _fstrcat(szErrorStatus, tmpstr);

        MessageBox(NULL, szErrorStatus, "Compile Error Message", MB_OK);
    }
    else {

        _fstrcpy(szLogFile, PERRORMSGINFO)lParam->szErrorStatus);
```

```
        hFile = OpenFile(szLogFile, &ofstruct);

        // process error log file...
    }

    break;
```

Note if multiple engines are open, the wParam parameter can be used to identify which engine has sent the error message.

### See Also

DSRegisterEngineMsgWnd

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_FUNCTIONERROR

WM_DSPENGINE_RUNTIMEERROR

### Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

# 5.2 Constant Definitions

There are numerous constants defined in DSPower series software include files, such as enmgr.h, hwmgr.h, hwlib.h, and splib.h. The sections below describe these constant definitions.